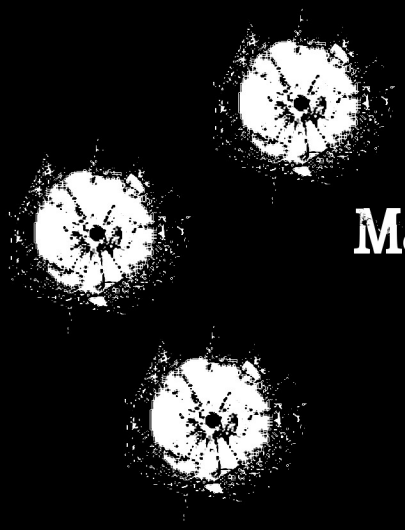
Terrorist's Revenge

Entwicklung, Programmierung und Design eines Ego-Shooters mit OpenGL



Maturaarbeit im Fach Informatik
von
David Halter (4mb)

betreuende Lehrkraft
Lukas Lippert

18. Oktober 2007

Inhaltsverzeichnis

Vorwort

15'000 Zeilen oder bei dieser Schriftgrösse 350 A4 Seiten Programmiercode sind das Resultat dieser Maturaarbeit. Ein extremer Aufwand. Die Zeit, die ich damit verbrachte, füllte auch einen grossen Teil meiner Freizeit in diesem Jahr. Vermutlich eine der aufwendigsten Maturaarbeiten überhaupt. Es gab viele Wochen, an denen ich über 20h am Programmieren war. Ja sogar Ferienwochen, an denen ich über 80h arbeitete.

Ausserdem kreierte ich über 100 Grafiken, einige 3D Modelle und Levels, die für das Spiel wichtig waren.

Die Idee für die Programmierung eines Spieles kam schon am Anfang des zweiten Jahres an der Kantonsschule auf. Das Ziel vieler Schüler war es immer wieder auf den Schulrechnern zu spielen. Da Spiele an der Kantonsschule grundsätzlich verboten waren, gab es immer wieder Diskussionen mit Herrn Brunner, dem Informatikadministrator, über den Sinn dieser Verbote. Dieser war zu dieser Zeit auch unser Physik- und Programmierlehrer. Er wich verständlicherweise nicht von seiner Position ab, jedoch sagte er, falls jemand ein Spiel selbst programmieren würde, wäre das Spielen erlaubt.

Gegen Ende des Programmierunterrichts fingen wir auch an mit OpenGL zu experimentieren. Dies führte mich dann zu der etwas utopischen Idee dies zu meiner Maturaarbeit zu machen. Als dann die Maturaarbeit anstand, fragte ich Herrn Lukas Lippert anfragte, ob er sich eine Zusammenarbeit vorstellen könnte.

Der Grundgedanke, warum ich einen Ego-Shooter und zum Beispiel nicht ein Strategiespiel geschrieben habe, war folgender: Shooter fand ich vor allem interessanter zu programmieren und man musste weniger Modelle entwerfen. Ausserdem sind Strategiespiele noch um einiges komplizierter und benötigen ein besseres Konzept. Wichtig ist in diesem Kontext auch, dass dieser Ego-Shooter sehr viele Eigenschaften von „Counter Strike“ übernommen hat, da ich so sehr leicht ein Konzept erarbeiten konnte. Bei einem Spiel wie diesem, kann man einfach einmal anfangen zu programmieren und immer weiter an den Details feilen.

Irgendwann taufte ich das Spiel dann auf den Namen „Terrorist's Revenge“, der auch beibehalten wurde.

Abstract

Mit Delphi und den Schnittstellen SDL, OpenGL und OpenAL wurde ein Spiel in der Form eines Ego-Shooters programmiert. Dieses Spiel trägt den Namen „Terrorist's Revenge". Für das Spiel wurde auch ein 2D-Editor (GUI-Editor) entwickelt, womit dann auch das eigene Design verwirklicht wurde. Im 3D Bereich, wurden Modelle und Levels mit 3ds Max entworfen und abgeändert und dann in das Spiel geladen. Die Programmierung erfolgte von Grund auf selbst, dabei wurden aber dennoch einige unabdingbare Bibliotheken verwendet.

Für das Spiel wurden eigene Netzwerkkomponenten programmiert, die Kollisionen wurden selbst berechnet, die Partikel selbst gezeichnet und so weiter. Neben den vielfach etwas älteren OpenGL Befehlen kamen auch neuartige Shader zum Zuge, die dann aber nicht so funktionierten wie gewollt.

1. Einleitung

Die Zielsetzung enthielt anfangs nur die Arbeit eines, auf die technischen Aspekte ausgerichteten, Spieles. Es war nicht das Ziel, ein möglichst ästhetisches Spiel zu erschaffen. Dies, weil ich keine Ahnung von der Erstellung von 2D und 3D Grafiken hatte und das Ausmass der Arbeit gesprengt wurde.

Jedoch fing ich dann mit der Zeit doch an, die Grafik wichtiger zu gewichten und änderte dann auch meine Zielsetzung zu einem Spiel um, dass sowohl grafisch und auch technisch etwas zu bieten hatte. Das hiess für mich konkret noch mehr Zeit zu investieren und einige kleinere Dinge in der Programmierarbeit zu streichen.

1.1. Stand der Forschung

Der Aufwand, der in der Computerspielindustrie betrieben wird, um die Spiele immer realistischer, aktionsreicher und detailreicher zu gestalten, ist enorm. Mir scheint, dass das Ziel vieler Entwickler und Studios sei, eine vollkommen realistische Welt zu erschaffen, in der man sich dann austoben könne. Viele Spieler sind grundsätzlich auch dieser Meinung und freuen sich über die technischen Innovationen. Dennoch sehen einige Spieler die Grafik eher als unwichtig und finden alte Spiele, durchaus reizender, als die neuen „überladenen" Spiele.

Diese Entwicklung der Spiele, ist aber auch stark der aktuellen Computertechnik unterstellt. Gerade auch neue Spieltitel müssen versuchen, ein möglichst breites Publikum zu erreichen. Dies wird dann immer durch massive Unterschiede der grafischen Qualität eines Spiels erreicht, die in den Optionen einstellbar sind.

Die Entwicklung des Computers ist vielfach auch auf die Spielindustrie zurückzuführen. Gerade Grafikkarten werden im Grossen und Ganzen fast ausschliesslich für die Gameprogrammierung entwickelt. Es gibt auch einige wenige Grafikkartenmodelle, die auf CAD-Gebrauch ausgerichtet sind oder sonstige industriellen Karten, aber zumindest die Grafikkarten in den Heimcomputern sind auf Spiele ausgerichtet.

Dennoch, die Grafikkarte ist noch längst nicht an ihrem Ende der Entwicklung angelangt. Gute Grafikkarten arbeiten mittlerweile mit bis zu einer Million Dreiecken[1], die aber dennoch zu wenig sind, für eine realistische Weltdarstellung. Spiele wie „Unreal Tournament 3" werden mit Charakter-modellen von zwei Millionen Dreiecken produziert[2], die dann heruntergerechnet werden müssen auf ein paar wenige tausend Dreiecke. Einige Outdoor-Szenen

1 http://www.delphigl.com/forum/viewtopic.php?p=55216#55216 , Stand 28. 9. 2007
2 http://www.unrealtechnology.com/html/technology/ue30.shtml : Distributed Computing Normal Map Generation Tool , Stand 28. 9. 2007

der „Unreal Engine" sind über 100 Millionen Dreiecke gross. Gezeichnet werden schlussendlich aber nur 500'000. Diese Selektion ist natürlich extrem wichtig und technisch gesehen enorm kompliziert.

Abbildung 1: Unreal Engine 3, eine Szene, die über 100 Mio. Dreiecke enthält

Es bleibt aber zu sagen, dass Grafikkarten viel weiter entwickelt sind, als die in etwa viermal höher getakteten CPUs. Die Stanford-Universität, erreichte einen Durchbruch durch die Entwicklung eines speziell angepassten Programms für die Berechnung der räumlichen Struktur der Aminosäuren[3]. Durch die parallel geschalteten Pixelshader, kann die Grafikkarte eine 20-40 mal höhere Rechengeschwindigkeit erreichen. Der Grund, wieso aber keine Grafikkarten als Rechenknechte verwendet werden, ist die schwerere Programmierung auf diesem Gebiet. Grafikkarten sind darauf ausgerichtet, Dreiecke zu zeichnen. Simple Dinge wie die Addition können zwar auch gerechnet werden auf der Grafikkarte, sie müssen aber erst gescheit transformiert werden, um auch eine effiziente und geschwinde Rechnung zu ermöglichen. Allerdings versucht man gerade auf dem Gebiet der CPU, Technologien der Grafikkarten zu verwenden und diese so zu verbessern.

3 http://www.heise.de/newsticker/meldung/78948 , Stand 28. 9. 2007

Eine weitere Innovation der aktuellen Forschung sind Physikkarten. Diese entlasten die CPU in all den extrem komplexen Interaktionen der Spielwelt. Die Physikkarten werden sich aber erst in einigen Jahren so richtig verkaufen. Die Spielstudios brauchen da noch einige Zeit, wie auch die Gamer selbst.

1.1.1. Kritiken

Die möglichst realistische Darstellung der Spiele brachte natürlich auch in der Vergangenheit heftige Kontroversen mit sich. Nach dem Attentat eines jungen Deutschen in Erfurt waren gerade in Deutschland die Politiker sehr schlecht auf Computerspiele zu sprechen. Jedoch konnten bisher eigentlich keine Studien klar erweisen, dass gewaltverherrlichende Spiele einen bemerkbaren Einfluss auf die Psyche haben. Im Gegenteil, eine kürzlich veröffentlichte Studie[4] erwies ein völlig anderes Bild des typischen Gamers, als jenes, dass er in der Gesellschaft hat.

Ein anderer Kritikpunkt ist die Suchtgefahr, vor allem bei MMORPGs (Massively Multiplayer Online Role-Playing Games) wie „World of Warcraft". Viele Spieler legen Spielzeiten von mehreren Stunden pro Tag hin und verlieren vielfach den Kontakt zur Realität, beziehungsweise zur Umwelt. Sie vernachlässigen ihre realen Freunde und suchen virtuelle Freundschaften. Aber auch da konnte nicht klar erwiesen werden, dass dieses Phänomen wirklich schädigend ist.

Gerade durch diesen Realitätsverlust gab es auch schon einen Todesfall in Korea[5]. Allerdings wird dieser Kritikpunkt von Spielern auch ein wenig belächelt, da dies ein Einzelfall war und ähnliche Fälle bisher noch nicht aufgetreten sind.

Selbst dazu Stellung nehmen kann und will ich nicht. Ich weiss nicht was für Auswirkungen Gewalt verherrlichende Computerspiele auf Menschen haben. Das Einzige was ich dazu sagen kann ist, dass mir selbst die kurze Zeit, in denen ich solche Spiele ab und zu gespielt habe, nicht wirklich geschadet hat. Es war für mich eine Art Hobby. Auf jeden Fall ist aber auch für Gamer wichtig, dass sie ab und zu mit realen Menschen in Kontakt kommen und auch einmal an die frische Luft gehen.

1.2. Theoretische Grundlagen

Ursprünglich war eigentlich mein Ziel, ein Spiel zu schreiben, welches nicht eine möglichst schöne Grafik vorweisen kann, sondern eine möglichst gute Engine (engl. Antrieb, Motor). Doch dazu später. In der 3D-Programmierung wird eigentlich immer ein Zugriff zur Grafikkarte benötigt. Diese „Zugriffe" nennt man in der Fachsprache auch Grafik-API (Grafik Application Programming Interface, engl. Für Programmierschnittstelle). Davon gibt es eigentlich nur zwei verschiedene Typen, OpenGL oder eben DirectX.

4 http://www.spiegel.de/netzwelt/web/0,1518,443473,00.html , Stand 28. 9. 2007
5 http://de.wikipedia.org/wiki/Computerspiel#Todesf.C3.A4lle , Stand 28.9. 2007

In der Spielprogrammierung genügt es aber eben nicht, nur eine Grafik-API zur Verfügung zu haben. Wichtig sind zum Beispiel auch die Fensterverwaltung, eine Möglichkeit Audio-Dateien abzuspielen oder eben auch die Bereitstellung einer Schnittstelle für Netzwerkzwecke. Normalerweise werden auch Physik-APIs verwendet . Doch zu all diesen Einzelteilen werde ich in späteren Teilen der Arbeit schreiben.

1.2.1. Die Engine

Die Engine ist eigentlich der Grundsatz jeglicher Gameprogrammierung. In allen grossen Spieltiteln werden Engines verwendet, für „Far Cry™" und das in der Zukunft erscheinende „Crysis™" zum Beispiel die „CryENGINE™"[6] oder für „Unreal Tournament 3" die „Unreal Engine 3"[7]. Die Unreal Engine wird und wurde von dutzenden Spielen verwendet. Die „Unreal Engine 2.5" wurde z.B. von 23 unterschiedlichen Spielen[8] verwendet.

Abbildung 2: Real-Life vs. Crysis

Das Schöne einer Engine ist, dass sie vielseitig verwendbar ist. Sie wird normalerweise so geschrieben, dass sie für Egoshooter, aber auch Simulationen brauchbar ist. Sie ist eigentlich ein Funktionswerkzeug für die Gamedesigner. Sie bietet Schnittstellen, um Dateien zu laden und leistet die ganze Optimierungsarbeit. Das heisst, die komplizierte Arbeit wird so dem Designer abgenommen und er kann sich darauf konzentrieren, seine Modelle so schön wie möglich zu gestalten und muss dabei keine Vorgaben bezüglich einer maximalen Anzahl Dreiecke einhalten. Ausserdem bieten Engines normalerweise Schnittstellen für Skriptsprachen, mit denen man dann die ganzen kleinen Details erledigen kann.

6 http://www.crytek.com/technology.html , Stand 28. 9. 2007
7 http://www.unrealtechnology.com/html/technology/ue30.shtml , Stand 28. 9. 2007
8 http://de.wikipedia.org/wiki/Unreal_Engine#Spiele_auf_Basis_der_Unreal_Engine , Stand 28. 9. 2007

Die Engine wird von den meisten Spielherstellern gekauft, um die Zeit besser einzuteilen und Kosten zu sparen (eine Engine zu entwickeln kostet sehr viel Geld). Es gibt allerdings auch einige nicht kommerzielle Engines, von denen die wohl bekannteste OGRE3D[9] ist. Diese Engine wurde von Anfang an Open Source entwickelt.. Ausserdem existiert die Quake-Engine[10], deren Versionen erst kommerziell verwendet wurden und immer erst einige Jahre später unter der GNU GPL[11] Lizenz herausgegeben wurden.

Abbildung 3: Iris 2, produziert mit OGRE3D

Eine Engine stellt alle Grundlagen zur Verfügung, von der simplen Bildausgabe, Schatten, Physik und vor allem auch komplizierte neuartige Shadersysteme.

9 http://ogre3d.org/ , Stand 30. 9. 2007
10 http://de.wikipedia.org/wiki/Quake-Engine , Stand 30. 9. 2007
11 http://de.wikipedia.org/wiki/GNU_GPL , Stand 30. 9. 2007

1.2.2. Polygone

Mit den Polygonen steht und fällt die Grafikprogrammierung. Ein Polygon (griech. Vieleck) wird durch eine Anzahl Punkte definiert, die durch einen Linienzug geschlossen miteinander verbunden werden. Dennoch spielen die Aussenlinien der Polygone eigentlich nie eine Rolle, Polygone werden immer gefüllt, mit:

1. einer Farbe,
2. mit je einer Farbe je Eckpunkt, wobei dann interpoliert wird,
3. mit Texturen, deren Koordinaten zwischen Eckpunkten interpoliert werden,
4. mit Kombinationen, aus den bisher genannten,
5. mit Shadern, wobei der Programmierer selbst bestimmen kann, wie die Farbe ausgegeben wird.

Polygone mit mehr als drei Ecken werden von der Grafikkarte durch Triangulation[12] automatisch in Dreiecke zerlegt. Da diese Unterteilung eigentlich immer Rechenleistung braucht, werden diese Polygone vom Programm normalerweise vor Laufzeit berechnet. Das heisst, es wird eigentlich immer mit Dreiecken gearbeitet.

Vielecke mit mehr als drei Punkten bergen einerseits den Nachteil, nicht auf einer Ebene zu liegen und somit nicht klar definierbar zu sein, andererseits gibt es das Problem der konkaven Polygone.

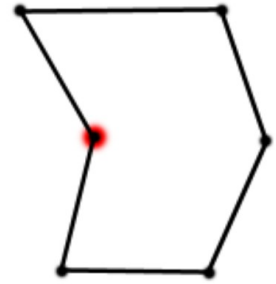


Abbildung 4: Polygon, konkav

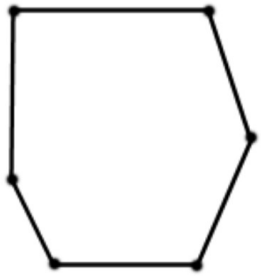


Abbildung 5: Polygon, konvex

Konkave Polygone können im Gegensatz zu konvexen Polygonen nicht einfach mit beliebigen Dreiecken durch Triangulation dargestellt werden. Das heisst, entweder ist ein grösserer Aufwand bei der Berechnung, was gezeichnet werden soll nötig oder das Polygon wird nicht richtig gerendert, sprich es wird konvex gezeichnet.

12 http://wiki.delphigl.com/index.php/Triangulation , Stand 30. 9. 2007

2. Werkzeuge & Hilfsmittel

Im Folgenden werde ich die Hilfsmittel und Werkzeuge auflisten, die bei meiner Arbeit benützt wurden. Allerdings werden einige kleinere Dinge weggelassen werden, da sonst der Umfang gesprengt würde.

2.1. Der Arbeitsplatz

Die ganze Arbeit ist eigentlich bis auf ein paar Ausnahmen in meinem Zimmer an einem, nun schon über drei Jahre alten „Targa Companion 811" Laptop entstanden. Design, Programmierung, wie auch der schriftliche Teil, wurden an diesem Notebook mehr oder minder erfolgreich abgeschlossen.

Ausserdem wurde ein neuwertiger Computer eingesetzt, der vor allem zu Netzwerk-Testzwecken diente. Auch die Informatikzimmer der Kantonsschule Frauenfeld wurden dazu „missbraucht", mein Programm zu testen.

2.2. Programme

2.2.1. Delphi

Beschreibung

Delphi ist eine IDE, die auf der Sprache PASCAL basiert und von Borland entwickelt wird. Delphi wurde kontinuierlich weiterentwickelt, wobei die aktuellste Version „Delphi 2006" ist. Die in den späten 1980er Jahren[13] sehr erfolgreiche Software „Turbo Pascal" wurde mittlerweile auch wieder in die Produktlinie Borland's aufgenommen und von Delphi her weiterentwickelt.

Delphi ist ein sehr anfängerfreundliches Programm, weshalb es auch vielfach von Schulen verwendet wird, was aber auch der leicht erlernbaren Sprache PASCAL zuzuschreiben ist.

Eigentlich wäre ich gerne auf Turbo Pascal umgestiegen, doch war es den Aufwand kaum mehr wert, sich noch einmal mit ähnlicher, aber doch ein wenig anderer Software zu beschäftigen.

Gebrauch

Circa 90% der Arbeitszeit dieses Projekts, widmete ich dem Programm Delphi. Wie auch im Anhang zu sehen, sind das riesige Textblöcke, die durch diese Software geschrieben wurden.

Zufrieden war ich im Endeffekt aber nicht mit meinem Delphi 2005. Einige Fehler führten immer wieder zu Abstürzen. Ausserdem fehlen einige Funktionen die in anderen IDEs standardmässig Inventar sind.

13 http://de.wikipedia.org/wiki/Turbo_Pascal , Stand 2.10.2007

2.2.2. 3ds Max

Beschreibung

3ds Max ist ein 3D-Grafik- und Animationsprogramm, das in professionellen Bereichen neben Maya vermutlich am meisten verwendet wird[14]. Es wurde von Autodesk entwickelt.

Das Programm ist für den Neueinsteiger ziemlich überladen und 99% des Funktionsumfangs wird er nie verwenden können. Allerdings kann der Anfänger vieles ignorieren und behält trotzdem den Überblick.

Abbildung 6: 3ds Max Logo

Gebrauch

Mit 3ds Max modelliert habe ich wenig. Im Internet habe ich die Modelle zusammengesucht und dann nur ein wenig mit 3ds Max abgeändert, um geeignete Modelle für mein Spiel zu haben.

Vielfach waren diese Änderungen auch nur Rotationen, bzw. Translationen, um die Objekte richtig laden zu können

2.2.3. GIMP

Beschreibung


Abbildung 7: Gimp Logo

GIMP bedeutet ***G****NU (GNU is Not Unix)* ***I****mage* ***M****anipulation* ***P****rogram* und ist, wie der englische Name schon sagt, ein Open-Source-Bildbearbeitungs-programm.

Der Schwerpunkt dieses Programms liegt auf der Bearbeitung einzelner Bilder. Durch viele Effekte und Arbeitsgeräte ist GIMP eine sehr gute Alternative zu kommerziellen Programmen wie zum Beispiel Photoshop.

Das Programm braucht wie auch viele andere Open-Source-Programme eine gewisse Zeit fürs Kennenlernen.

Gebrauch

GIMP war wohl das Programm, das ich nach Delphi am meisten verwendet habe. Mit GIMP entstanden alle selbst gemachten Grafiken. Besonders hervorzuheben ist dabei das selbst gemachte GUI-Thema.

14 http://de.wikipedia.org/wiki/3ds_max#f.C3.BCr_Filmproduktionen , Stand 2.10.2007

2.2.4. OpenOffice.org

Beschreibung

OpenOffice.org ist ein freies und quelloffenes Office-Paket. Der direkte Konkurrent ist das Paket „Microsoft Office". OpenOffice beinhaltet Programme für Tabellenkalkulation, Textverarbeitung, Präsentation, ein Zeichenprogramm, ein Formeleditor, sowie auch ein Datenbankprogramm.

Abbildung 8: OpenOffice.org Logo

Gebrauch

OpenOffice wurde von mir vor allem zum Schreiben dieser Arbeit verwendet. Ausserdem wurde es ab und zu für Kalkulationen verwendet, um die Arbeit beim Programmieren zu erleichtern.

Ich benütze OpenOffice und nicht Microsofts Office, da diese durchaus vergleichbar sind und OpenOffice als Freeware auch gefördert werden sollte.

2.3. Schnittstellen/DLLs

2.3.1. OpenGL

Beschreibung

In der Wikipedia, wird OpenGL so definiert:

> „**OpenGL** (**Open G**raphics **L**ibrary) ist eine Spezifikation für ein plattform- und programmiersprachenunabhängiges API (Application Programming Interface) zur Entwicklung von 3D-Computergrafik. Der OpenGL-Standard beschreibt etwa 250 Befehle, die die Darstellung komplexer 3D-Szenen in Echtzeit erlauben. Zudem können andere Organisationen (zumeist Hersteller von Grafikkarten) proprietäre Erweiterungen definieren."[15]

OpenGL wird von vielen verschiedenen Programmiersprachen implementiert. Im Gegensatz zu Microsofts DirectX ist OpenGL plattformunabhängig und somit die einzige Möglichkeit, Games auf Linux zu spielen (ohne Emulator).

15 http://de.wikipedia.org/wiki/OpenGL Stand 2.10.2007

Gebrauch

Abbildung 9: OpenGL Logo

Zum Gebrauch von OpenGL gibt es eigentlich nicht viel zu sagen, da OpenGL in meinem Projekt quasi allgegenwärtig ist. Ohne OpenGL wäre meine Arbeit mehr oder weniger sinnlos.

Interessant ist lediglich, dass eigentlich nur selten OpenGL-Befehle in meinem Quelltext zu sichten sind. Das liegt aber vor allem daran, dass die meisten Befehle andauernd wiederholt werden und somit nicht so klar sichtbar sind.

2.3.2. OpenAL

Beschreibung

Abbildung 10: OpenAL Logo

OpenAL (**Open A**udio **L**ibrary) kann und soll als Ergänzung zu OpenGL gesehen werden. OpenAL weist auch sehr viele Ähnlichkeiten mit OpenGL auf. Die Befehle sind sehr ähnlich aufgebaut, Namenskonventionen von OpenGL wurden erhalten. Der Programmierer kann so eine Symbiose der beiden Schnittstellen erreichen und hat es leichter zu programmieren.

OpenAL ist sehr leicht portierbar und hat so eine relativ grosse Verbreitung erreicht.

Auch hier hat OpenAL einen direkten Konkurrenten. Dieser nennt sich DirectSound und ist auch Bestandteil von DirectX.

Gebrauch

OpenAL wurde für das gesamte Audiosystem verwendet. Allerdings wurde der grösste Teil der OpenAL Programmierung nicht von mir übernommen, sondern stammt von M. van der Honing[16].

16 http://www.noeska.com/en.aspx , Stand 2.10.2007

2.3.3. SDL

Beschreibung

SDL (Simple DirectMedia Layer) ist eine freie, plattformunabhängige Multimedia-Bibliothek. SDL wird auch im kommerziellen Bereich vielfach verwendet um Spiele von Windows nach Linux zu portieren, vor allem durch Loki Games[17].

Auch für viele Programmiersprachen ist SDL vorhanden, da das Potential von SDL wahrgenommen wurde.

Abbildung 11: SDL Logo

Gebrauch

SDL wurde von mir vor allem für die Ereignis- und Fensterverwaltung verwendet. Allerdings habe ich viele Funktionen von SDL vernachlässigt, wie zum Beispiel das Laden und Speichern von Dateien.

2.3.4. SDL_Net

Beschreibung

SDL_Net ist ein Teil von SDL und kann zur Netzwerkprogrammierung verwendet werden. SDL_Net bietet Anbindungen, an die Netzwerkprotokolle TCP und UDP.

Gebrauch

SDL_Net wurde bei mir für die ganze Netzwerkprogrammierung verwendet. Der Grund, warum ich SDL_Net und nicht irgendwelche ähnlichen Komponenten genommen habe, war vor allem, dass beide Protokolle gegeben waren. Ausserdem war die Plattformunabhängigkeit schön.

2.4. Bibliotheken

Ich liste im Folgenden nur Bibliotheken auf, die ich auch wirklich gebraucht habe. Es gab durchaus Bibliotheken, wovon ich nur zwei bis drei Befehle gebraucht habe, diese werde ich hier nicht extra dokumentieren.

17 http://de.wikipedia.org/wiki/Loki_Software , Stand 2.10.2007

2.4.1. gl3ds.pas

Beschreibung

Die gl3ds.pas wurde von M. van der Honing[18] entwickelt und ist ein Kernelement meiner Programmierarbeit. Diese Unit lädt 3D Modelle und bietet gerade auch Möglichkeiten an, diese Modelle anschliessend zu rendern.

Ausserdem kann man die Vertexdaten wieder aus dieser Unit auslesen.

Gebrauch

Diese Unit war extrem wichtig für meine Arbeit. Durch diese Bibliothek wurde mir sehr viel Arbeit abgenommen und ich konnte mich auf anderes konzentrieren.

Die Charaktermodelle werden alle durch die gl3ds.pas gezeichnet. Die Welt wird durch gl3ds geladen und dann wiederum von mir ausgelesen und weiterverarbeitet. So fällt der Aufwand fürs Laden der Dateien weg.

2.4.2. textures.pas

Beschreibung

Die textures.pas wurde von Jan Horn[19] 2001-2002 entwickelt. Sie enthält Methoden um JPG, BMP und TGA Dateien zu laden. Sie ist wohl eine der simpelsten Bibliotheken in diesem Bereich. Viele andere Bibliotheken, die auf das Laden von Texturen ausgerichtet sind, sind objektorientiert und massiv komplizierter.

Gebrauch

Die textures.pas benütze ich für alle Bilder, die ich selbst lade und nicht von einer externen Bibliothek, wie zum Beispiel gl3ds.pas. Ich habe mich für diese Unit entschieden, da viele andere Units in diesem Bereich meiner Meinung nach relativ überladen sind und für meine Zwecke zu viel des Guten.

18 http://www.noeska.com/dogl/gl3ds.aspx , Stand 2.10.2007
19 http://www.sulaco.co.za/ , Stand 2.10.2007

2.4.3. dglOpenGL.pas

Beschreibung

Die dglOpenGL.pas ist eine Bibliothek, durch die man auf OpenGL-Befehle zurückgreifen kann. Sie wird von der deutschen Online-Community „delphigl.com" entwickelt und bietet Support für alle möglichen OpenGL-Extensions.

Gebrauch

Ich verwende diese Unit für alle möglichen OpenGL-Befehle. Sie wird im Gegensatz, zu vielen anderen Bibliotheken in diesem Bereich dauerhaft weiterentwickelt und läuft sehr stabil.

2.4.4. glFrustum.pas

Beschreibung

glFrustum.pas wurde von Sascha Willems[20] entwickelt und hat den Zweck, zu bestimmen, ob ein Punkt, eine Kugel, oder eben auch ein Quader im Bereich der Kamera liegt.

Gebrauch

Diese Unit ist vor allem für den Octree wichtig. Dort wird bestimmt ob ein Teil dieses Octrees gerendert wird. So kann man massiv Rechenleistung sparen, da man eben die Objekte, beziehungsweise die Polygone, welche nicht auf der Kamera gesehen werden können, ignoriert.

2.4.5. Delphi Bibliotheken

Delphi bringt natürlich selbst auch einige Bibliotheken mit sich. Zur standardmässig implementierten „system.pas" gesellen sich die plattformunabhängigen „messages.pas", „dialogs.pas" oder eben auch „sysutils.pas".

Viele dieser Bibliotheken werden nicht sehr intensiv gebraucht und beinhalten viele Standardbefehle. Von der „sysutils.pas" verwende ich vielfach das Kommando „inttostr", ein sehr simpler Befehl.

Die Unit „windows.pas" habe ich persönlich nicht verwendet, allerdings braucht die Unit „3ds.pas" die Windows-Unit. Was das für die Plattformunabhängigkeit bedeutet werde ich später erklären.

20 http://saschawillems.de/ Stand 2.10.2007

3. Vorgehen

3.1. Übersicht

Ich habe, entgegen der Erwartungen vieler Leute, keine Zeitplanung gemacht. Die Zeitplanung lief eigentlich in meinem Kopf ab und war viel mehr etwas in der Richtung: „Wenn die Zeit reicht, dann mache ich noch dies und jenes." Ich habe mir einige wenige Ziele gesetzt, die ich dann mehr oder weniger locker einhalten konnte.

Vieles habe ich einfach mit „Todo"-Listen gemacht. Ich habe mir aufgeschrieben, was ich noch machen muss. Diese Listen sind vereinzelt auch in den einzelnen Units zu finden und haben vielfach noch optionale Ziele, die ich teilweise nicht erfüllt habe.

3.2. Zeiteinteilung

Ich dokumentiere je Kapitel ein Quartal und versuche dabei immer in etwa anzugeben, was ich hauptsächlich gemacht habe. Allerdings ist dies meistens nicht wirklich transparent. Vielfach wurde noch nebenbei an kleineren Dingen geschrieben. Units wie der Eventhandler oder auch die Kamera entstanden nebenher.

Ich werde auch in Fussnoten zu meinem Blog (quasi eine Dokumentation, die ich während der Arbeit gemacht habe)[21] verweisen, wo Interessierte auch nachlesen können, was denn genauer passiert ist. Mit Details will ich in dieser Arbeit verschonen, zumal Vieles auch nicht so wirklich interessant oder sehr kompliziert ist.

3.2.1. bis Herbst 06

Die Zeit bis Herbst 07 war eigentlich überhaupt nicht strikt geplant. Es lag zwar ein erster Prototyp vor, den ich aber dann zu grossen Teilen wieder verworfen habe. Das Einzige, was noch von dieser Zeit stammt, ist die Partikelengine. Diese jedoch ist völlig veraltet und hätte dringend eine Generalüberholung nötig. Units wie die Movement und Cam-Unit haben aus dieser Zeit ihre Namen und auch kleine Codefetzen beibehalten.

21 http://terrorists-revenge.blogspot.com/ , Stand 5.10.2007

3.2.2. Herbst 06 bis Winter 06/07

Vor den Herbstferien 07 wurde dann bekannt gegeben, dass eine Maturaarbeit zu schreiben sei, als Bedingung für eine Matura. Dies veranlasste mich dann dazu das Thema fest zu setzen und ich überarbeitete als erstes die GUI[22]. Dies dauerte etwa 3 Monate[23]. In dieser Zeit ist dann auch der Hauptteil dieser Unit entstanden. Mit der Zeit kam dann auch einmal ein erster[24] GUI-Editor heraus.

3.2.3. Winter 06/07 bis Frühling 07

Zwischen den Weihnachts- und Sportferien wurde dann mit Herrn Lippert eine Zusammenarbeit für die Maturaarbeit abgemacht. Ich habe ihm dann auch eine Liste mit den Punkten, die ich erfüllen wollte, geschickt. Diese beinhaltete die Zielsetzung im Bereich der verschiedenen Units. Darin wurde aber nicht sehr viel festgehalten und ich war relativ frei von meiner Arbeit her. Das einzig erwähnenswerte war das Ziel, nach den Sommerferien mit dem Schreiben der Maturaarbeit anzufangen.

Danach habe ich noch ein bisschen an der GUI weiter geschrieben, die dann bald schon fertig war[25], um dann möglichst schnell auch Levels laden zu können. Dies war ein sehr wichtiger Punkt, denn durch das schnelle Schreiben dieses Teils konnte ich mir selbst sehr viel Stress ersparen.[26]

22 http://terrorists-revenge.blogspot.com/2006/11/gui-coding.html , Stand 5.10.2007
23 http://terrorists-revenge.blogspot.com/2006/12/gui-goes-10.html Stand 5.10.2007
24 http://terrorists-revenge.blogspot.com/2007/01/gui-editor.html , Stand 5.10.2007
25 http://terrorists-revenge.blogspot.com/2007/03/gui-goes-final.html , Stand 5.10.2007
26 http://terrorists-revenge.blogspot.com/2007/03/levels.html , Stand 5.10.2007

3.2.4. Frühling 07 bis Sommer 07

Erst versuchte ich mich mit Shadern. Als ich mich dann eingelesen hatte, schrieb ich auch die ersten Shader, ich scheiterte jedoch am eigenen PC, der nicht fähig war, etwas bessere Shader darzustellen. Damit war dieses (zeitaufwändige) Thema vorerst gestrichen.

Nebenbei entwickelte ich das Netzwerk und Kollisionen[27]. Dies lief soweit sehr gut. Nicht zuletzt dank meinem Vater, konnte ich die Kollisionen fürs erste als abgeschlossen betrachten und einen sehr gut optimierten Code vorweisen.

Die Netzwerkprogrammierung lief auch reibungslos. Probleme kamen erst im nächsten Quartal auf.

Nachdem ich dann einen grossen Teil der Programmierarbeit abgeschlossen hatte, begann ich, vermehrt dem Design Beachtung zu schenken, welcher eigentlich nicht in der Zielsetzung der Arbeit vorhanden war. Aber mein Gedanke dahinter war, dass eine Engine selbst nicht wirklich viel nützt. Sie muss auch demonstriert werden können. Somit verbrachte ich dann viel Zeit mit GIMP, um Bilder zu entwickeln.

3.2.5. Sommer 07 bis Abgabe am 22. Oktober

Da ich sowohl im Design, wie auch in der Programmierung an einem Punkt gekommen war, den man zumindest ohne die Fehlerausmerzung als fertig bezeichnen könnte, arbeitete ich nur noch relativ wenig. Des Weiteren liess die Motivation etwas nach, da ich doch seit fast einem Jahr sehr viel an diesem Projekt gesessen bin und Fehlerbekämpfung einfach etwas sehr Nervenaufreibendes und Langsames ist. Nebenbei fing auch die Vorbereitung, für die Wintersaison im Volleyball an, was mir zusätzlich Zeit raubte, bei 4 Trainings pro Woche. Somit hatte ich bis etwa Anfang Herbstferien nur ein bisschen getestet und nicht wirklich viel gemacht. Dann begann ich wieder sehr intensiv zu arbeiten, um sowohl die Arbeit zu schreiben, wie auch die letzten grossen Fehler – vor allem im Netzwerk – zu beseitigen.

27 http://terrorists-revenge.blogspot.com/2007/05/shadernetworkkollisionen.html Stand 5.10.2007

4. Dokumentation

4.1. Units

Die Units werden im Folgenden immer ohne die Endung „.pas" verwendet. Man kann auch die Unterschiede zwischen neueren und älteren Units erkennen – die neueren Units beginnen jeweils mit einem U. Hier werden nur selbst programmierte Units dokumentiert.

Die folgenden Units sind auch die Kernstücke meiner Programmierarbeit. Die Quelltexte befinden sich im Anhang.

Ich werde versuchen, mich relativ kurz zu halten und nur das wichtigste zu dokumentieren. In vielen Maturaarbeiten werden die einzelnen Funktionen analysiert und dokumentiert. Ich hingegen werde nur oberflächlich die Unit ansprechen, da ansonsten alleine dieses Unterkapitel einige 100 Seiten einnehmen würde.

Es werden nur Units des Spiels dokumentiert, der GUI-Editor wird später dokumentiert.

4.1.1. Basics

Die Basics-Unit enthält vor allem verschiedene Vektortypen. In ihr sind auch relativ viele geometrische Operationen untergebracht (Skalarprodukt etc.). Wichtig sind aber vor allem die Kollisionsfunktionen. Wie diese entstanden sind, beziehungsweise wie diese Prozeduren nun arbeiten, wird im Kapitel „5.1 Kollisionen" genauer eingegangen.

4.1.2. CamUnit

Die CamUnit ist, wie der Name schon sagt, die Heimat der Berechnungen für die Kamera. Die Kamera wird hier an den richtigen Ort gesetzt und richtig ausgerichtet. Zur Zeit werden mit dieser Unit vier verschiedene Kameras unterstützt, die man im Spiel jeweils mit F1 wechseln kann.

4.1.3. EventHandlerUnit

Einerseits findet die Koordination von Tastatur und Keyboard Ereignissen statt, sowie auch die Zeitverwaltung. Hier wird zum Beispiel alle 3 Sekunden den Mitspielern gesendet, wie es um ihre Lebenspunkte steht.

Andererseits ist hier auch der „PacketCollector" stationiert, der sich darum kümmert, wann TCP Packete gesendet werden.

4.1.4. FileTypes

FileTypes ist eine sehr simple Unit. Sie hat den Sinn, Dateien leichter auslesen und beschreiben zu können. Ausserdem wäre diese Unit die perfekte Vorbereitung, um auch vom Dateisystem her plattformunabhängig zu werden. Ich hätte mit relativ kleinem Aufwand SDL_RWops[28] einfügen können, was plattformunabhängig wäre. Die Zeit dazu hatte ich allerdings bisher nicht. Es sind auch zwei verschiedene Typen vorhanden, binär und ASCII. Das letztere wurde aber nie gebraucht.

4.1.5. gameVARUnit

Diese Unit lädt die Einstellungen und übernimmt gleichzeitig auch das Verarbeiten, wenn Änderungen dieser Einstellungen vorliegen (während der Laufzeit). Allerdings wurde ein Grossteil dieser möglichen Einstellungen noch nicht konkret verwirklicht und diese Unit liegt in einem sehr schönen objektorientierten Rohformat da.

28 http://wiki.delphigl.com/index.php/Tutorial_SDL_RWops , Stand 4.10.2007

4.1.6. GUI

Die GUI (Game User Interface) ist eine eigenständige Unit, die auch ohne das Spiel benutzt werden kann. Durch die GUI wird versucht, dem Benutzer das Leben leicht zu machen. Durch Beschreibungen und Texte wird geholfen. Die GUI ist seltsamerweise die Unit, wo ich weitaus am meisten Arbeit hineingesteckt habe. Sie umfasst ca. 5000 Zeilen und somit über ein Drittel des Gesamtcodes. Die extreme Arbeit, die darin steckt, ist aber nicht auf die Wichtigkeit dieser Unit zurückzuführen, sondern vielmehr auf die Faszination an der Objektorientierung, die ich während der Programmierung dieses Arbeitsteiles bekommen habe. Ein fortgeschrittener Programmierer wird aber auch merken, dass vieles sehr unsauber programmiert ist. Aber die GUI läuft sehr gut und ich bin mit der Arbeit sehr zufrieden.

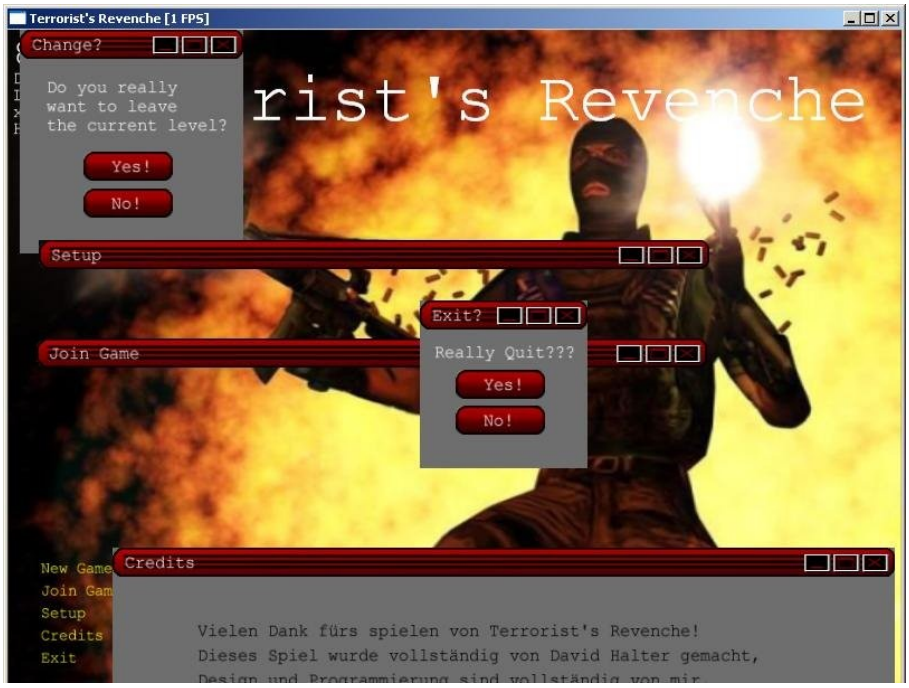


Abbildung 12: GUI, im Spiel, zu früheren Zeiten

Die GUI umfasst, soviel ich weiss, 16 verschiedene Komponenten - von Buttons, Fenstern, bis zu Editfeldern. Sie ist also sehr gross und besitzt auch einen grossen Funktionsumfang für Programmierer.

4.1.7. GUIAdds

GUIAdds ist der Name für Zusätze zur GUI. Diese Teile sind aber an das Spiel gebunden. Die Aktionen werden hier koordiniert und auf Veränderungen wird reagiert. Ausserdem ist hier die Konsole untergebracht, ein relativ wichtiges Werkzeug, um Fehlerquellen rasch zu eliminieren.

4.1.8. Mainunit

Der Code der Mainunit wurde vermutlich am meisten verändert. Immer wieder wurde der Text total neu geschrieben. Die Unit ist so aufgebaut, dass sie möglichst wenig machen muss. Es wird eigentlich fast immer auf andere Units verwiesen und selbst gerechnet wird eigentlich nichts.

Sie ist relativ sauber programmiert und auch gut dokumentiert.

Es entstanden aber trotzdem hier die meisten Fehler, da sich ein Vertauschen von Zeilen grauenhaft auswirkt und zu stundenlanger Fehlersuche zwingen kann.

4.1.9. MovementUnit

Die MovementUnit ist für die Kollisionen und die Fortbewegung der Figuren verantwortlich. Durch die Bewegung der Maus und die gedrückten Tasten, wird ermittelt, in welche Richtung sich der Charakter bewegt.

4.1.10. oooal

Die oooal – Unit wurde zu 90% von M. van der Honing[29] geschrieben. Sie ist sehr schön und vorbildlich objektorientiert programmiert und darum auch leicht nutzbar. Sie lädt leider lediglich .wav Formate, was für den Programmierer nicht gerade praktisch ist, da dieses Format im Vergleich zu mp3 Formaten in etwa zehn mal mehr Platz braucht.

Diese Unit basiert auf OpenAL und ist darum auch plattformunabhängig.

29 http://www.noeska.com/doal/ , Stand 4.10. 2007

4.1.11. PartikelUnit

Die PartikelUnit ist uralt, ungefähr 1.5 Jahre. Sie ist dementsprechend auch ziemlich schlecht und nicht wirklich brauchbar. Sie ist der einzige komplette Überbleibsel aus der Zeit, vor dem Anfang der Arbeit. Eine Neu-programmierung würde eigentlich anstehen, doch die Zeit liess dies nicht zu.

Was sie weiterhin schlecht macht ist, dass die dazugehörigen Units „pfxCore" und „pfxImp" nicht von mir selbst geschrieben wurden und nur zu einem kleinen Teil geändert wurden. Diese selbst sind Demonstration eines Tutorials[30].

4.1.12. UCharacters

Die UCharacters – Unit ist vermutlich die Wichtigste im ganzen Spiel. Sie enthält einerseits die Verwaltung der Waffen und andererseits die Verwaltung der Charaktere. Somit ist sie neben der Unit „Ulevels" eigentlich hauptsächlich für die graphische Darstellung des Spiels verantwortlich.

In diese Unit fallen auch Dinge wie zum Beispiel der Respawn, Schüsse oder eben die Kommunikation mit der Netzwerkunit.

4.1.13. ULevels

ULevels ist eigentlich nur ein Zwischenstück zwischen der Mainunit und UOctree. Diese Unit lädt die „.3ds" Dateien und wandelt diese dann in „.lvl" um. Ausserdem werden hier die „Skyboxen" geladen, also die Bilder für den Himmel und Horizont.

Das schöne an dieser Unit ist vor allem, dass beliebige 3D Objekte aus 3ds Max geladen werden können und dann darin gespielt werden kann. Somit kann 3ds Max als Leveleditor verwendet werden und es müssen keine externen Programme verwendet werden.

4.1.14. UNetwork

Die Netzwerk-Unit beinhaltet die Kommunikation zwischen Server und Client. Die Autorisierung und die wichtigen Daten werden dabei über das TCP Protokoll, die eher unwichtigen Daten, welche mehrmals in der Sekunde verschickt werden, benützen UDP.

Mit dem Netzwerk hatte ich so einigen Ärger und noch nicht alles scheint behoben zu sein.

4.1.15. Unit_SDL

Auch diese Unit entstand mit einem Tutorial[31] und gehört zum älteren Kaliber.

30 http://wiki.delphigl.com/index.php/Tutorial_Partikel1 , Stand 4.10.2007
31 http://wiki.delphigl.com/index.php/Tutorial_SDL_Einstieg , Stand 4.10.2007

Sie enthält die De- und Initialisierung von SDL, wie auch OpenGL.

4.1.16. UOctree

UOctree ist eine Unit, die unter anderem mit einem Tutorial[32] über Octrees entstanden ist. Sie enthält auch die Grundlage für eine Rendertechnik mit „OcclusionQuery“[33]. Dies wäre eine relativ schnelle Rendertechnik, allerdings wurde sie (noch) nicht umgesetzt.

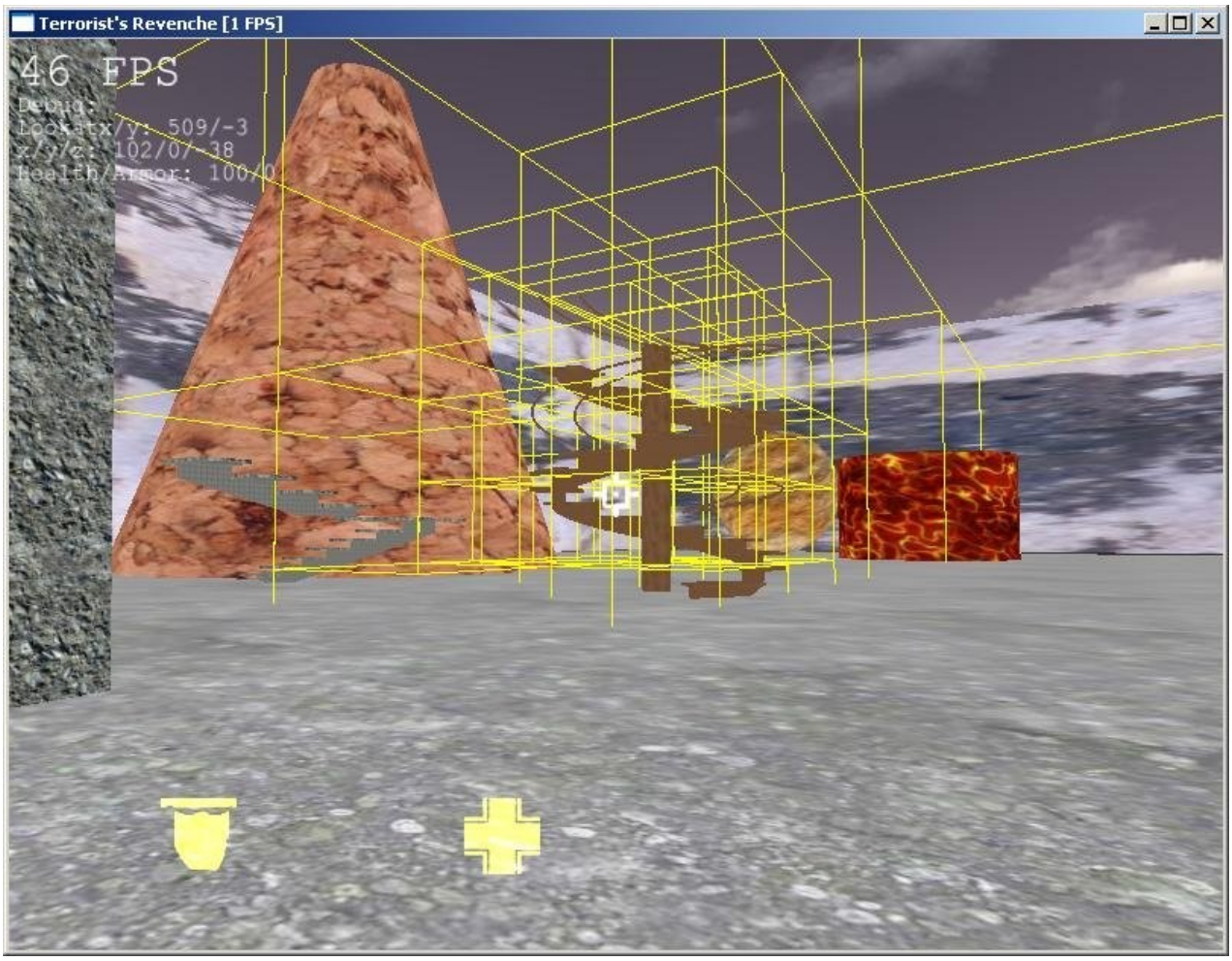


Abbildung 13: Ein Level, inkl. den "Nodes"

Das schöne eines Octrees ist, dass er sich rekursiv (das heisst eine Prozedur ruft sich selbst auf) immer wieder in einzelne Nodes spaltet. Dies geschieht je nach Anzahl Dreiecken, die in einem gewissen Raum vorhanden sind. Wie man unten im Bild sieht, geschieht dies bei der Treppe sehr oft, da an dieser Stelle viele Dreiecke vorhanden sind.

32 http://wiki.delphigl.com/index.php/Tutorial_Octree , Stand 4.10.2007
33 http://wiki.delphigl.com/index.php/Tutorial_NVOcclusionQuery , Stand 4.10.2007

4.1.17. UShader

Die Shader – Unit wurde eigentlich dafür entwickelt um auch neuartiger OpenGL Technologie eine Chance zu geben. Die Shader werden Objekt-orientiert geladen. Das Ganze hat allerdings nicht so funktioniert wie ich mir dies vorgestellt habe, was ich aber im Kapitel „Probleme" schildern werde.

4.2. Der GUI-Editor

Der GUI-Editor wurde entwickelt, um schnell und einfach die 2D-Grafik für das Spiel zu gestalten. Er ist auch sehr praktisch aufgebaut und besitzt eine eigene ca. 1000 Zeilen (28 Seiten, bei dieser Schriftgrösse) grosse Unit, die mit der GUI-Unit aus dem Spiel kommuniziert.

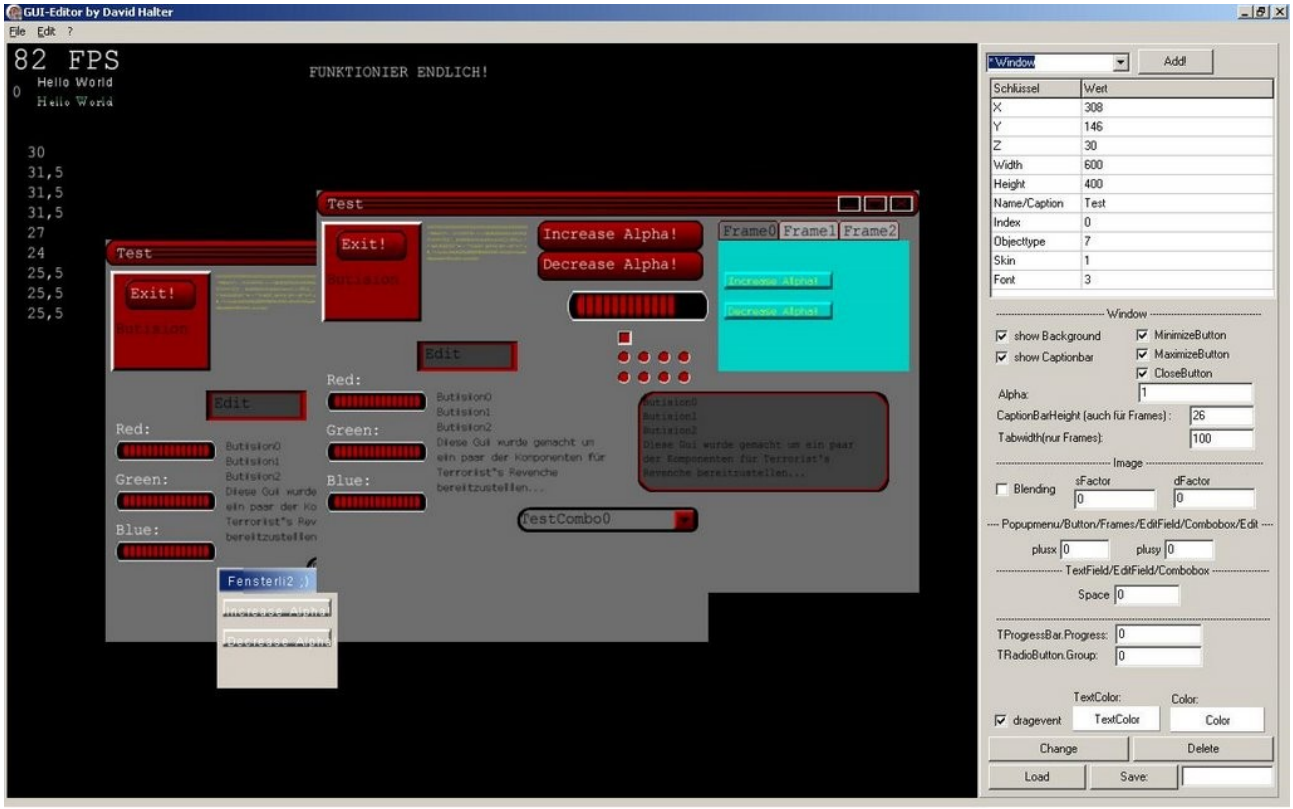


Abbildung 14: Der GUI-Editor

5. Probleme

Ich schildere im folgenden wie ich mit den Hauptproblemen umgegangen bin und wie ich sie gelöst habe.

Es gab natürlich noch viele andere Probleme. Das waren aber im Grossen und Ganzen immer Programmierfehler, die ich selbst verursacht hatte. Solche Probleme konnten locker 10 Stunden Programmierarbeit einnehmen, aber interessant sind sie nicht.

Nur um einmal zwei kleine Probleme zu schildern:

1. Ich habe einmal 10h Sucharbeit leisten müssen, nur um herauszufinden, dass die Datei „SDL.dll" defekt war.
2. Vielfach habe ich zwei Zeilen vertauscht, die dann zu Fehlern führten. Mit solchen Dingen habe ich auch sehr viel Zeit verloren.

Aber nun zu den „grossen" Problemen.

5.1. Kollisionen

Die Kollisionen waren vermutlich die komplizierteste, aber gleichzeitig auch interessanteste Sache. Darum werde ich darauf auch genauer eingehen. Die Programmierung der Kollisionen benötigte jede Menge Vektorgeometrie. Es wurden auch verschiedene Arten von Kollisionsabfragen gebraucht. Auf diese Kollisionen möchte ich nun genauer eingehen.

5.1.1. Kollision der Spielerfigur mit der Welt

Die Spielerfigur kann vereinfacht auch durch eine Kugel dargestellt werden. Dies wird gemacht, um eine höhere Performance zu erreichen. Kollisionen müssen so nicht mit einer sehr komplexen Spielerfigur verrechnet werden, sondern können mit einer Kugel, die durch das Zentrum und den Radius definiert ist, gemacht werden.

Die effektive Kollision mit der Welt wird dann durch eine rekursive Funktion verursacht. Sie erfolgt rekursiv, durch die einzelnen „Nodes" des „Octrees". Dabei wird noch eine Prüffunktion vorgeschoben, die prüft ob sich die der Mensch überhaupt in einem „Node" befindet. Da das ganze rekursiv geschrieben ist, werden somit auch die jeweiligen „Kinder" vernachlässigt und es kann sehr viel Zeit gespart werden.

Danach folgt die Kollision der Kugel mit den Dreiecken. Es wird geprüft, ob ein Dreieck die Kugel schneidet. Die Implementation dieser Kollision war eine sehr komplizierte Sache. Mit meinem Vater zusammen, wurden mehrere Seiten für

die Herleitung geschrieben. Erst wird geprüft ob die Ebene des Dreiecks die Kugel schneidet. Falls dies zutrifft, wird das Zentrum der Kugel senkrecht auf die Ebene projiziert und geprüft ob der Punkt im Dreieck liegt. Das Problem dieser Projektion ist aber, dass gewisse Dreiecke, die die Kugel nur streifen, vernachlässigt werden. Da dies aber eher wenige und unwichtige Fälle sind, reicht diese Approximation.

Die Prüfung ob ein Punkt in einem Dreieck liegt, ist wiederum sehr schnell. In diesem Bereich gibt es viele Methoden[3435] wobei diese ist wohl eher eine komplizierte und unbekannte, aber braucht einfach wenig Rechenleistung. Der Trick der Methode ist das Erstellen eines zufälligen Strahls. Bei diesem wird dann geschaut, wie viele Kollisionen er mit den Kanten des Polygons erreicht. Was dazu führt, dass bei einer ungeraden Anzahl Kollisionen der Punkt innerhalb des Polygons liegt, bei einer geraden Anzahl ausserhalb.

Somit hätte man schon eine Kollision mit der Welt, man kann also nicht mehr überall hindurch laufen. Allerdings reicht dies für ein 3D Spiel nicht aus, da alle Bewegungen auch in vertikaler Richtung gemacht werden möchten.

Steigungen/Treppen

Die Steigungen und Treppen zu überwinden, war nach einer guten Idee relativ einfach. Ich prüfte erst, ob eine Kollision mit der Welt statt fand. Falls ja, setzte ich den Menschen senkrecht auf das Polygon. Natürlich musste dann noch einmal eine Prüfung mit der Welt erfolgen, damit man nicht eine Treppe hinaufsteigen konnte und gleichzeitig durch die anliegende Wand ginge.

Gravitation

Die Gravitation wurde mit Hilfe von Geschwindigkeiten errechnet. Ich beschleunigte den Menschen immer und schaute, ob der Mensch mit der Umwelt kollidiere. Wenn dies der Fall war, setzte ich den Menschen zurück und die Geschwindigkeit auf Null. Durch die Einstellung der Oberflächenbeschleunigung (g) auf kleine Werte, konnte ich auch durch die Welt fliegen.

5.1.2. Schüsse

Die Schüsse sind natürlich essentiell in einem Ego-Shooter. Sie waren auch nicht so schwer zu programmieren, wie die Kollisionen mit der Welt. Zuerst wurde geprüft, ob der Schuss den Mensch trifft, danach ob der Schuss auch noch Polygone schneidet. Dabei kam mir die Programmierung, der Polygon/Punkt Kollision zugute, da diese wieder verwendet werden konnte.

34 http://en.wikipedia.org/wiki/Wikipedia:Reference_desk/Archives/Mathematics/2006_December_6 , Stand 18.10.2007
35 http://www1.acm.org/pubs/tog/editors/erich/ptinpoly/ , Stand 18.10.2007

5.2. Netzwerk

Das Netzwerk war am Anfang sehr leicht und ohne Probleme zu programmieren. Die Programmierung ging schnell vorwärts und bereitete keine Probleme. Als dann aber die ersten Testversuche gemacht wurden, bereitete das Netzwerk riesige Probleme.

Das Ganze wäre nicht so schlimm gewesen, wenn irgendjemand eine Ahnung von SDL_Net (TCP & UDP) unter Delphi gehabt hätte und mir Beihilfe geleistet hätte. Bis ich nur schon einzelne Pakete empfangen konnte, ging es sehr lange.

Ein weiteres grosses Problem der Netzwerkprogrammierung ist, dass man alle Daten synchronisieren muss und zwar mit geringer Netzwerkbelastung. Die Bewegungen wurden über UDP gesendet, dass in etwa 3 Mal schneller ist als TCP. Alles andere wurde über TCP gelöst.

Doch da stellte sich das nächste Problem: Bei jeder Sendung eines Pakets mit TCP stoppte der Server für kurze Zeit. Dies liess sich dann mit dem einfügen von einem anderen Thread ändern. Somit musste ich mich auch noch kurzfristig in Multithreading einarbeiten, um das Netzwerk problemlos laufen zu lassen.

5.3. Geeignete Modelle finden

Da ich nicht selbst modellieren konnte und auch keine Zeit dazu hatte es zu lernen, war die Modellsuche mein allergrösstes Problem. Durch die Entdeckung von http://www.turbosquid.com/ , konnte ich mir dennoch einige brauchbare Modelle besorgen, die gratis zur Verfügung standen und auch relativ schön gemacht waren.

Alle Waffen sind grundsätzlich von Turbosquid, viele wurden aber noch nachträglich bearbeitet, einige sogar massiv. Auch die Charaktermodelle sind von Turbosquid, diese wurden aber doch alle massiv abgeändert.

Bei den Modellen änderte ich vor allem die Texturen. So konnte ich neue Kleidung erzeugen, ohne einen riesigen Aufwand zu haben.

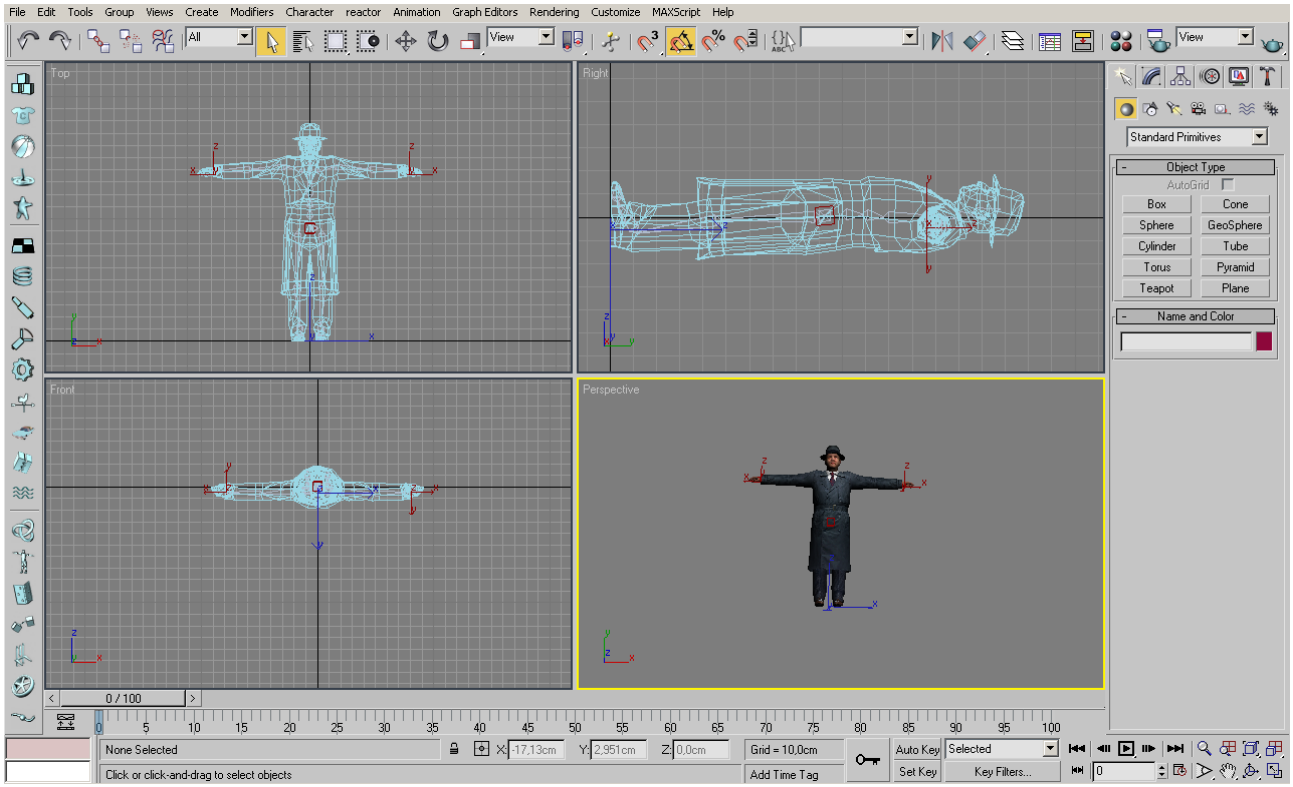


Abbildung 15: 3ds Max, am bearbeiten eines Charakters

5.4. Die Plattformunabhängigkeit

Die Plattformunabhängigkeit war keine Zielsetzung in diesem Spiel, sondern viel mehr eine Option, das Ganze auch für Linux brauchbar zu machen. Allerdings merkte ich dann nach einiger Zeit, dass es relativ schwer ist, auch so zu programmieren. Die Benutzung von SDL, OpenGL und OpenAL, war eben nur der Anfang. Viel mehr mussten viele Details angepasst werden, einige Bibliotheken durften nicht verwendet werden. Das schlimmste aber an allem war – ich keine Ahnung von Linux hatte, fast noch schlimmer, ich hatte nicht einmal eine installierte Linux Distribution.

Diese Dinge führten dann dazu, dass zwar ein relativ grosser Teil meines Codes plattformunabhängig gebraucht werden könnte, schlussendlich aber das Dateisystem für Windows gebaut worden ist. Darum wurde diese optionale Möglichkeit nicht erfüllt.

5.5. Shader

Shader sind neuartige Technologien in der Grafikprogrammierung. Mit einem Shader kann man bestimmte Pfade, die von den Treibern sonst fix bestimmt sind, selbst programmieren.

Es war also schliesslich auch mein Ziel, diese Technologie einmal kennen zu lernen und auch damit zu arbeiten. Shader kann man aber nicht mit PASCAL beziehungsweise Delphi schreiben. Shader können nur in einer C++ ähnlichen Sprache geschrieben werden, die auch eigene Typen und Funktionen kennt. Somit musste ich erst einmal diese neue Sprache kennen lernen. Dies geschah mit einem ca. 40 Seiten langem Tutorial[36].

Mein Ziel war es dann irgendwann einmal HDR-Effekte (High Dynamic Range Effekte) zu erzeugen.

Abbildung 16: Vergleich HDR und LDR am Spiel Half Life 2

HDR ist im Vergleich zu LDR (Low Dynamic Range) nicht realistisch. Ein Bildschirm bringt nur einen gewissen Kontrast zustande. Viele Computerbildschirme haben einen Kontrast von 1:1000. Gute Plasmabildschirme bis hin zu 1:10'000. Ein gesundes Auge aber nimmt einen Kontrast von 1:1'000'000 wahr. HDR versucht nun vor allem, die sehr hellen Lichter klarer erscheinen zu lassen. Generell helle Flecken werden sehr hell, Punkte der mittleren Helligkeit eher ein bisschen dunkler.

Ich habe dann erst den „Gaussian Blur" Effekt (Weichzeichnen) verwendet, um dann das Ganze zu verrechnen. Dies hat zwar sehr gut funktioniert, allerdings war die Performance derart miserabel, dass ich sicher nie mit Shadern spielen kann.

36 http://wiki.delphigl.com/index.php/Tutorial_glsl , Stand 5.10.2007

Zwischendurch gab es dann auch wieder sehr amüsante Erlebnisse mit Shadern. Wie unten im Bild ersichtlich, kann man durch Shader ein Bild sehr schön manipulieren.

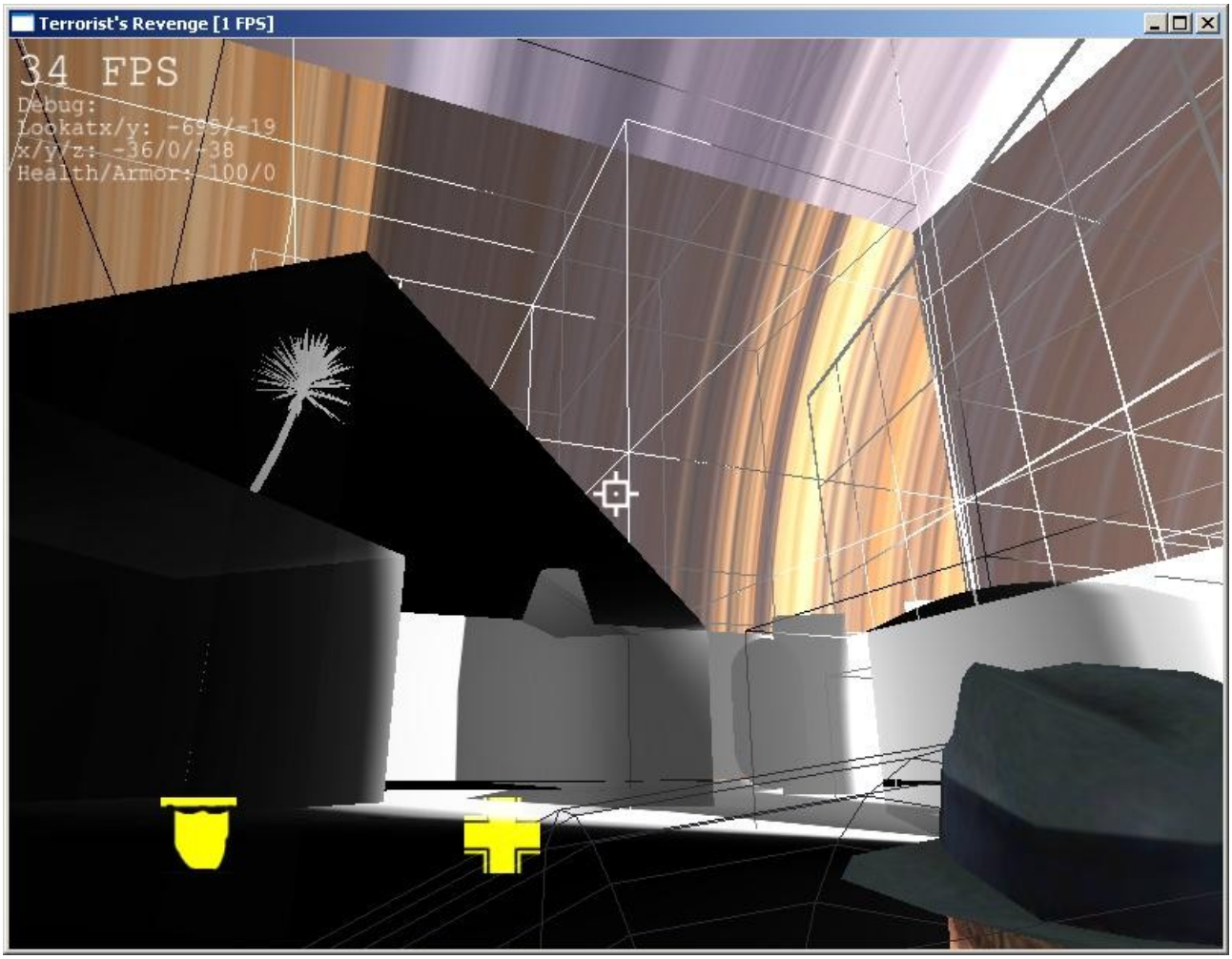


Abbildung 17: Ein etwas misslungener Shader-Versuch

6. Resultate

Das Resultat ist ein Spiel namens „Terrorist's Revenge", dass für ein Amateurprojekt durchaus gut aussieht und über Netzwerk spielbar ist. Ich glaube doch, bisher genug über das Projekt und den Hergang geschrieben zu haben. Darum werde ich nun vor allem die Bilder sprechen lassen.

Abbildung 18: Startbildschirm mit GUI

Die Programmierung der GUI war durchaus wichtig und auch sehr praktisch. So konnte ich sehr leicht meine Ideen umsetzen.

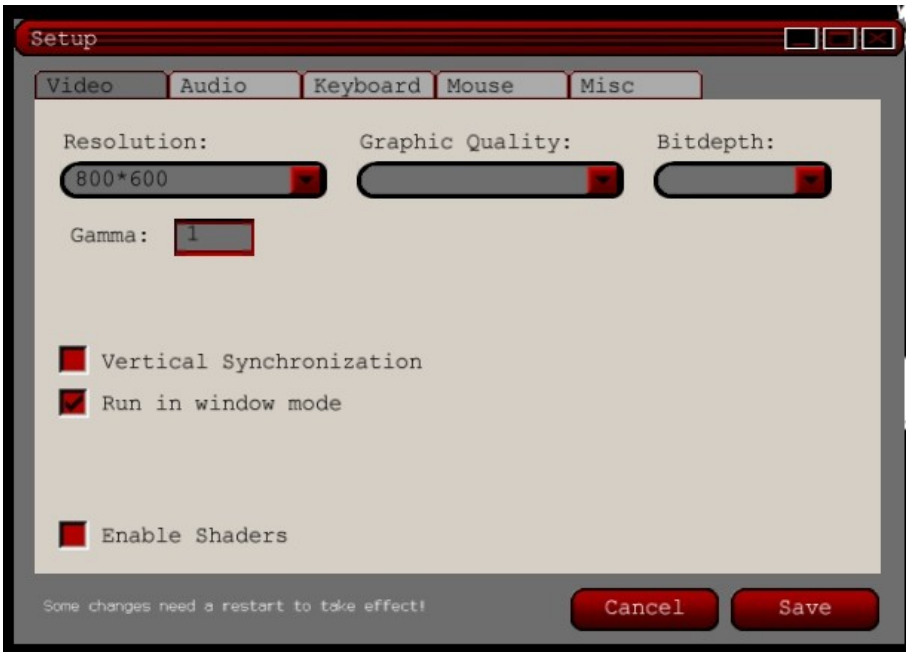


Abbildung 19: Setup

Dazu gehörte auch das Setup Fenster, welches meiner Meinung nach sehr gut gelungen ist. Implementiert sind nicht alle Einstellungen, aber gut aussehen tut es. Im unten stehenden Bild wird auch die Vielfalt der GUI gut ersichtlich.

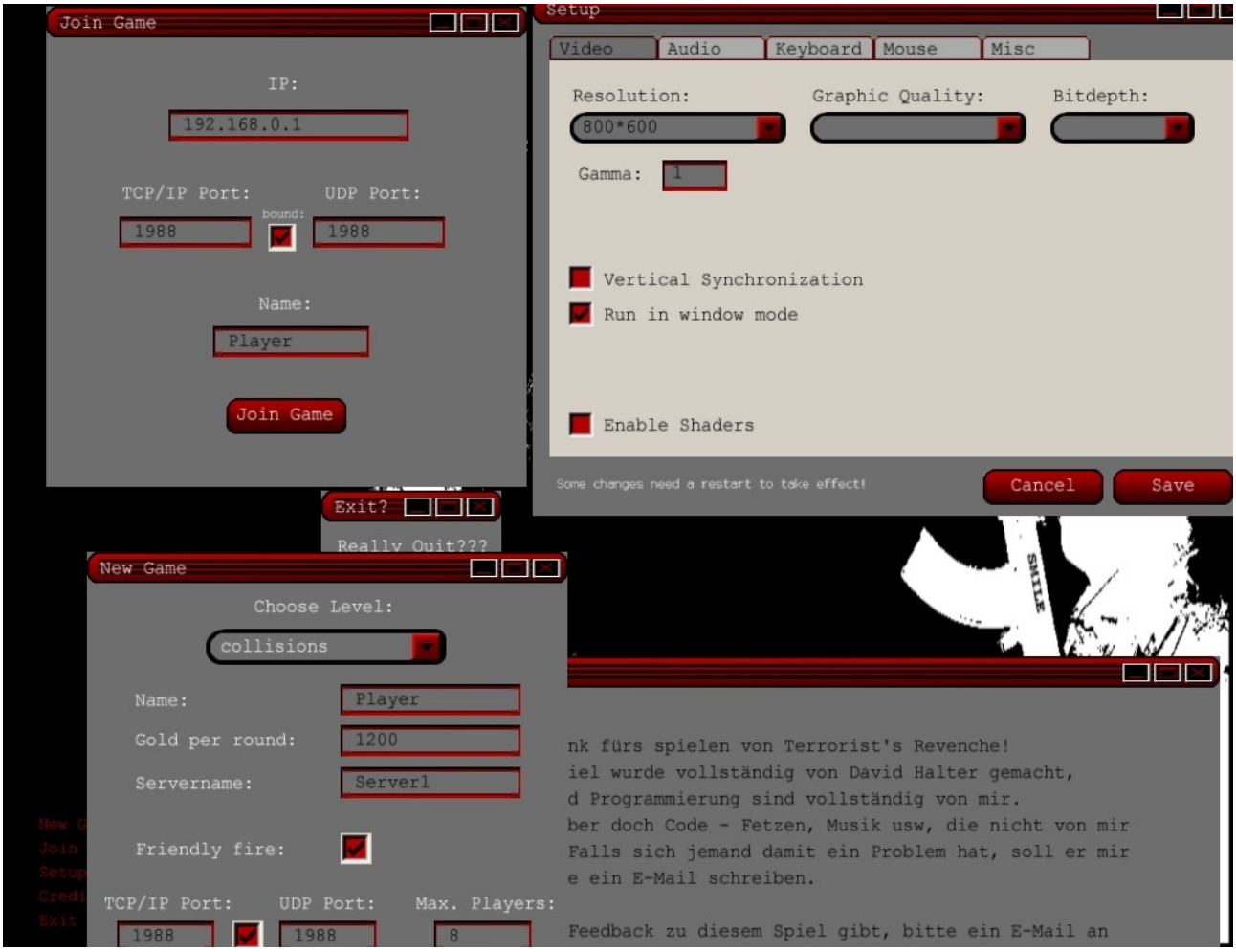


Abbildung 20: Einige GUI-Fenster zusammen

Abbildung 21: Ingame, Respawn

Abbildung 22: Ingame, Chat

Auch im Spiel war die GUI wichtig, für verschiedene Meldungen oder Fenster, wie zum Beispiel den Chat. Auch andere

Abbildung 23: Das Spiel mit einem Testlevel

Das Spiel kann sehr vielfältig gestaltet werden. Durch das laden aus 3ds Max, können viele Objekte geladen werden. Dieses Level umfasst ca. 30'000 Polygone.

Auch Partikeleffekte, wie zum Beispiel dieses Feuer sind möglich. Diese Feuereffekte sind im Spiel aber eher selten anzutreffen, da sie nicht standardmässig bei den Levels geladen werden.

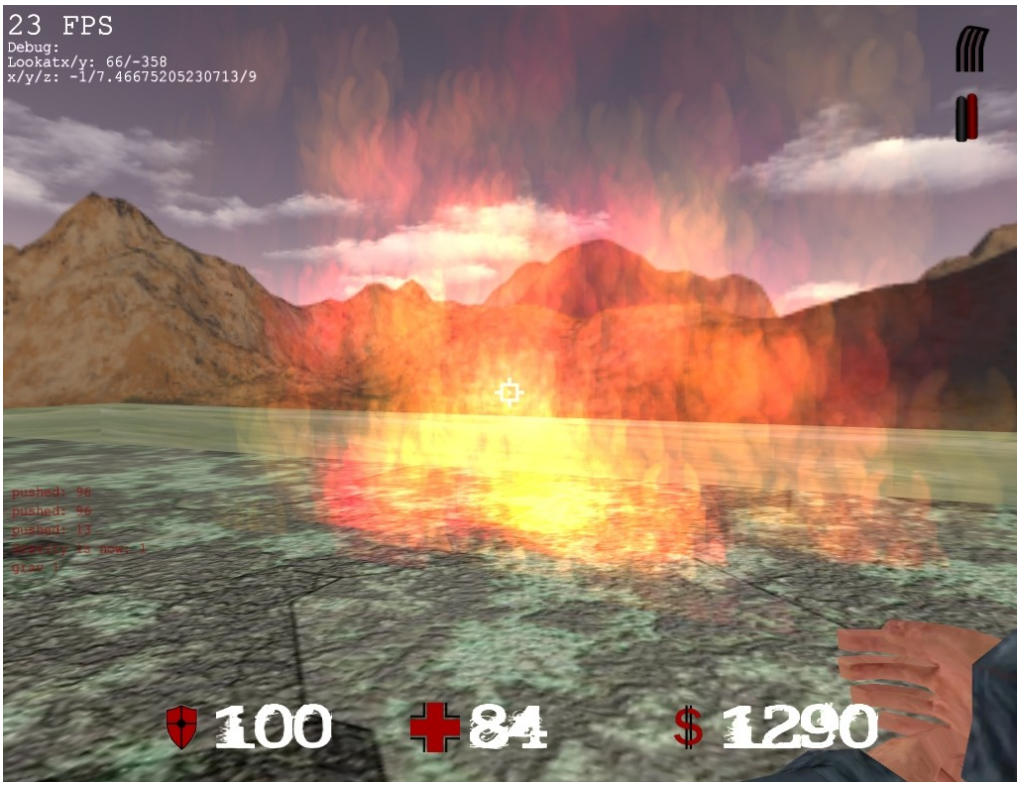


Abbildung 24: Im Spiel, Feuereffekte

Abbildung 25: Ingame, Modellauswahl

Abbildung 26: Ingame, Statistiken

Im Spiel wird auch eine Modellauswahl zur Verfügung gestellt, wie auch Statistiken. Die Statistik wird über das Netzwerk geführt und beinhaltet alle Spieler.

Alle Informationen werden im Spiel über das Netzwerk ausgetauscht und mogeln ist so eher schlecht möglich.

Abbildung 27: Spiderman meets Terrorist's Revenge

Durch die Einstellung der Oberflächenbeschleunigung kann der Spieler durch die Lüfte schweifen. Möglich ist es auch, die Geschwindigkeit des Spielers zu verändern.

Im Buy-Menu können 10 verschiedene Waffen und mehrere Ausrüstungsgegenstände gekauft werden.

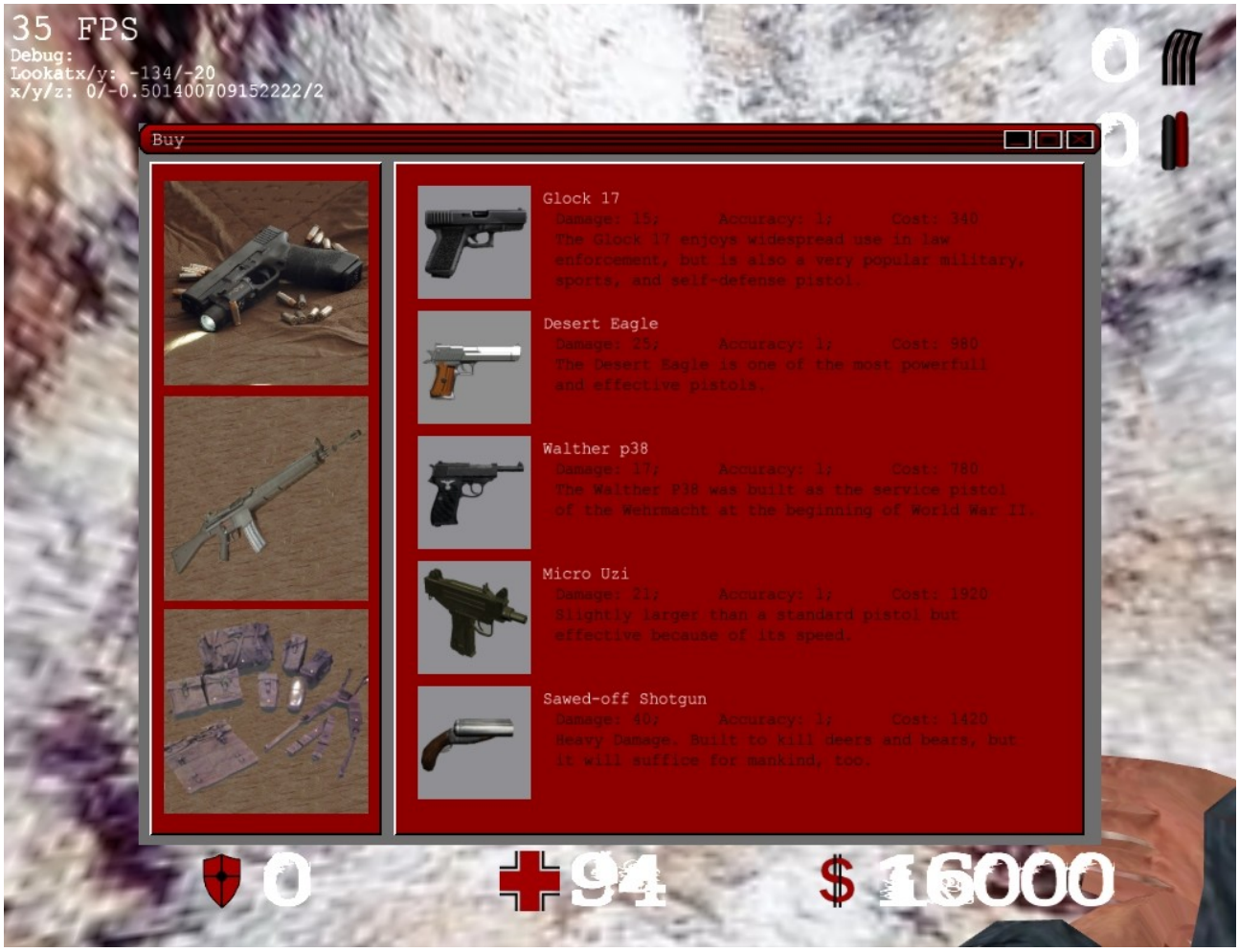


Abbildung 28: Das Buy-Menu

7. Diskussion

7.1. Wozu kann das Entwickelte eingesetzt werden

Das Entwickelte ist im Allgemeinen nicht wirklich brauchbar. Vieles wurde explizit für dieses Spiel entwickelt und ist somit kaum noch brauchbar. Allerdings kann ich natürlich auch in der Zukunft, falls ich wieder einmal OpenGL mit Delphi benötigen sollte, gewisse Codefetzen verwenden, die bestimmt sehr praktisch sein werden.

Eine Ausnahme bildet die GUI, inklusive GUI-Editor. Sie wurde völlig frei entwickelt und ist eigenständig. Das kann für die OpenGL-Programmierung natürlich sehr praktisch sein, da man eigentlich immer grafische Oberflächen benötigt, um ein Spiel attraktiver zu machen und eine leichte Handhabung zu garantieren.

7.2. Erfahrungen bei der Programmierung

Die Erfahrungen, die ich während der Programmierung dieses Spiels gesammelt habe, sind enorm. Ich habe nicht nur mit OpenGL eine Plattform kennen gelernt, die für 3D Animation praktisch ist, vielmehr habe ich gelernt sauber und objektorientiert zu programmieren, immer mit einem Auge auf die Performance. Meine Programmierkentnisse waren vor dieser Arbeit sehr beschränkt. Mittlerweile sind diese schon auf einem guten Niveau angekommen. Ich habe auch gemerkt, wie leicht es sein kann, sich in neue Programmiersprachen einzuarbeiten. Es ist nur noch eine Arbeit von einigen Stunden, bis man ähnliche Programmiersprachen einigermassen beherrscht. Dies kann wirklich sehr praktisch sein für ein Informatikstudium, welches ich eigentlich zur Zeit anstrebe.

7.3. Erfahrungen beim Design

Auch beim Gestalten dieses Projektes habe ich einiges hinzugelernt. Ich beherrsche nun GIMP und gewissermassen auch 3ds Max. Ich habe auch erneut gesehen, wie schwer es ist, geeigneten Inhalt selbst zu kreieren. Es ist eine sehr langwierige Arbeit, weswegen man sich auch nicht wundern muss, dass bei grossen Spielen dutzende Grafiker angestellt sind.

8. Schlussfolgerung und Ausblick

Die Programmierung von „Terrorist's Revenge" war sicherlich eine hochinteressante und spannende Arbeit. Aber wenn man den Aufwand mit einbezieht, hat sich das Ganze überhaupt nicht gelohnt. Für mich war die ganze Arbeit einfach ein Hobby und nicht Aufwand, den ich für die Schule betrieben habe.

Gerade der riesige Zeitaufwand macht es mir leicht, eine Schlussfolgerung zu ziehen. Die Programmierung eines Spieles dieser Grössenordnung ist schlichtweg zu viel für eine Maturaarbeit. Ich habe es nun durchgezogen, aber ich möchte allen Schülern empfehlen, falls sie sich für die Grafik-programmierung interessieren, kleinere Demos oder eben Spiele wie Solitär in 3D zu programmieren und zum Beispiel Effekte dazu zu basteln.

Wichtig zu sehen ist in diesem Kontext eben auch, dass diese Arbeit eigentlich durch Autodidaktik erfolgt ist. Für SDL zum Beispiel gibt es zwar Literatur, aber nicht etwa Literatur für SDL mit Delphi. Die Literatur richtet sich normalerweise nach der Programmiersprache „C++". Das heisst konkret, wenn man irgendwo Probleme hat, kann einem niemand helfen. Als ich beispielsweise die UDP Komponenten mit SDL_Net programmiert habe, konnte mir auch in Foren niemand helfen, da dies praktisch noch nie gebraucht wurde unter Delphi.

Terrorist's Revenge wird höchstwahrscheinlich nicht weiterentwickelt werden, da ich in der Vergangenheit einfach die Lust verloren habe, noch mehr meiner Freizeit zu investieren. Anfangs mag das Programmieren ja sehr spannend sein, aber wenn sich dann die nervigen Fehler einschleichen, dann ist schnell einmal Schluss mit der Motivation. Ich werde aber dennoch versuchen, Terrorist's Revenge noch im Internet zu veröffentlichen und Feedback zu kriegen, um einmal zu erkennen, wie mein Spiel denn so auf Spieler wirkt.

Das Spiel kommt natürlich nicht an kommerzielle Spieltitel heran und es steht somit ausser Frage, es irgendwie kommerziell zu veröffentlichen. Es wird im Internet zum freien Download verfügbar sein.

Gelernt habe ich aber dennoch sehr viel und ich bin auch froh so eine grosse Arbeit abgeschlossen zu haben. Ich habe natürlich insofern auch für ein allfälliges Informatikstudium viel Wissen angeeignet, was ansonsten sehr viel Zeit braucht. Gerade mit grundlegenden Programmiertechnischen Dingen, wie Pointern oder auch der Objektorientierung kenne ich mich nun sehr gut aus.

9. Zusammenfassung

Mit dieser Arbeit entstanden 15'000 Zeilen Programmiercode (ca. 350 Seiten), über 100 Grafiken und einige 3D Modelle/Levels. Das Ziel dieser Maturaarbeit war es, ein Spiel zu erstellen, das vor allem auf die (programmier-)technischen Aspekte ausgerichtet war. Irgendwann wurde dann diese Zielsetzung über den Haufen geworfen und das Design wurde wichtiger.

Die Programmierarbeit wurde mit Delphi erledigt, das auf der Sprache PASCAL basiert. Verschiedene Schnittstellen wurden verwendet, von denen vor allem OpenGL wichtig ist, da diese direkt auf die Grafikkarte zugreift und eine leichte 3D-Programmierung ermöglicht. Nebenbei wurden auch noch SDL und OpenAL verwendet, eine Multimedia- und eine Audioschnittstelle.

Am Anfang wurde vor allem für die benutzerfreundliche Oberfläche programmiert, womit dann auch ein 2D-Editor (GUI-Editor) entstand. Dadurch konnte ich mein eigenes Design verwirklichen. Grafiken entstanden vielfach auch mit GIMP.

Für den 3D Bereich wurde 3Ds Max verwendet. Dieses sehr leistungsfähige und komplexe Programm benötigte ich nicht nur, um die Charaktere zu verändern und gestalten, sondern vielmehr auch um die einzelnen Levels zu erschaffen.

Das netzwerkfähige Spiel operiert mit mehreren Protokollen, UDP und TCP. Bis das Spiel einmal über Netzwerk spielbar war, verging sehr viel Zeit und fast ein Vierteljahr verfloss mit der Netzwerkprogrammierung.

Auch die ganze Physik im Spiel wurde selbst programmiert. Kollisionen und Schüsse wurden berechnet. Dies geschah vor allem auch mit einem Blick auf die Performance.

Weitere eigene Systeme, die entwickelt wurden, sind Partikel, die Kamera oder eben auch die neuartigen Shader, die aber nicht so funktionierten wie gewollt. Die Performance wurde durch die Shader extrem herunter gezogen und nicht einmal moderne Rechnersysteme konnten diese, mit einer für das Auge angenehme Frequenz, darstellen.

Das Spiel wurde irgendwann einmal auf den Namen „Terrorist's Revenge“ getauft.

Die selbst entwickelte Engine kann übrigens in keinem einzigen Punkt mit modernen Engines, wie zum Beispiel der „Unreal Engine 3“ mithalten. Die Engine lässt mit viel älteren Engines vergleichen und schneidet auch da eher schlecht ab, da für die Gameprogrammierung einfach ein viel grösserer Aufwand erforderlich ist.

10. Dank

Sehr viele Personen haben etwas dazu beigetragen, dass diese Arbeit schlussendlich so herausgekommen ist. Zuerst einmal möchte ich der Online-Community von „delphigl.com" danken. Ohne diese Internetseite, hätte ich wohl überhaupt keine Chance gehabt, in diesem Projekt vorwärts zu kommen. Gerade auch die Tutorials im Wiki[37] waren sehr viel Wert und mit diesen kam ich auch schnell vorwärts.

Einen Dank geht in diesem Zusammenhang auch an alle OpenGL Programmierer im Netz, die mit Delphi/PASCAL operieren und so eine gewisse Basis und Akzeptanz vor allem für Delphi, aber auch OpenGL geschaffen haben.

Des weiteren möchte ich meiner Familie danken, vor allem meinem Vater, der mir bei den Kollisionen massiv geholfen hat. Aber auch meiner Mutter herzlichen Dank, für die Korrekturlesung.

Herrn Lippert, meinem Betreuer, möchte ich natürlich auch danken. Er hat durch sein Interesse an diesem vielleicht eher etwas utopischem Projekt, einiges beigetragen und auch eine gewisse Motivation meinerseits aufrecht erhalten.

Zuletzt möchte ich dem Herrn im Himmel danken, der mir sicher auch immer beigestanden hat während dem ganzen Projekt.

37 www.wiki.delphigl.com , Stand 5.10.2007

11. Literaturverzeichnis

Literatur in Form von Büchern wurde nicht verwendet. Fast alles wurde selbst hergeleitet und geschrieben. Webseiten wurden aber dennoch einige zur Hilfe gebraucht:

- www.delphigl.org inkl. Forum, Stand 18.10.2007
- www.wiki.delphigl.org , Stand 18.10.2007
- http://www.pascalgamedevelopment.com/ , Stand 18.10.2007
- www.de.wikipedia.org/ , Stand 18.10.2007
- http://www.turbosquid.com/ , Stand 18.10.2007
- http://www.gamasutra.com/ , Stand 18.10.2007
- diverse andere Webseiten

12. Glossar

API	Application Programming Interface, englisch für ->Schnittstelle
DLL	Dynamic Link Library, eine Datei, die als Schnittstelle gilt
Extensions (OpenGL)	Zusätze zu OpenGL, die neue Effekte und Funktionen ermöglichen
Frustum	Der Bereich, den man in der Kamera sieht
GLSL/GLSlang	Eine Sprache, um die ->Renderpipline umzuprogrammieren
IDE	Integrated Development Environment, engl. für die Integrierte Entwicklungsumgebung
IP	ein Protokoll in der Netzwerktechnik
Lightmaps	Texturen, die Informationen über die Lichter wiedergeben
Normale	Von einer Ebene senkrecht abstehender Vektor
Objekt-orientierung	Ein Ansatz der Programmierung um verschiedene Operationen und Eigenschaften zu klassifizieren.
Octree	Eine Datenstruktur, bei der die Knoten immer acht oder keine Nachfolger haben
Open-Source	Der Code dieser Programme ist frei einsehbar, darf aber nicht unbedingt auch weiterverwendet werden
PASCAL	Eine einfach zu verstehende Programmiersprache
Pixel	Ein einzelner Punkt auf dem Bildschirm, Bildschirme haben davon normalerweise über 1 Million.
Plattformun-abhängigkeit	Ein Programm kann auf mehreren Plattformen (das heisst Mac, Linux, oder Windows) abgespielt werden
Pointer	Engl. für Zeiger, man „zeigt" auf gewisse Punkte im Arbeitsspeicher
Renderpipline	Auf der Grafikkarte, die strikte Abfolge, wie ein Bild auf den Bildschirm gebracht wird
Schnittstelle	Werden von anderen Softwaresystemen zur Verfügung gestellt, um auf ihre Funktionen zugreifen zu können
Skript	Wird während der Laufzeit eines Programms geladen
TCP	Ein Netzwerkprotokoll, wobei die Daten ankommen
UDP	Ein Netzwerkprotokoll, bei dem Daten blind verschickt werden
Unit	In Delphi ein Teil bzw. eine eigenständige Bibliothek der Arbeit

13. Anhang

Der Anhang ist für diese Maturaarbeit schlichtweg zu gross und befindet sich in einem separaten Bund.